

Memoria del proyecto ETPA2

Memoria

1Dam G2

**Rui Qian,David Dobeson,Victor Martinez**

Programación y Base de Datos

8/06/2025



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Versión | Fecha | Motivo de modificación | Elaboración | Revisión | Aprobación |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Introducción

**El presente documento detalla la implementación de "ETPA 2", un sistema de gestión de elementos para un juego de rol (RPG) desarrollado en Java. El proyecto se enfoca en la interacción con una base de datos MySQL para la persistencia de datos relacionados con usuarios, personajes, mochilas y objetos, así como la incorporación de funcionalidades de inicio de sesión, registro y presentación narrativa.**

**Este proyecto representa una aplicación práctica de los conocimientos adquiridos en el primer año de DAM, abarcando conceptos fundamentales de programación orientada a objetos (POO), gestión de bases de datos relacionales y diseño de software modular.**

**Para el diseño y la conceptualización de "ETPA 2", se emplearon diversos diagramas que sirvieron como guía para la estructura del juego y la arquitectura del sistema. Estos incluyen un diagrama de flujo narrativo, un diagrama de clases UML y un diagrama entidad-relación (E-R), los cuales se describen en detalle más adelante.**

2. Arquitectura del Proyecto

**El proyecto sigue una estructura modular, organizada en paquetes que promueven la separación de responsabilidades, lo que facilita el mantenimiento y la escalabilidad del código. La arquitectura se compone principalmente de las siguientes capas:**

* **es.nebrija.controller: Esta capa contiene la lógica principal de la aplicación (Main.java), actuando como el punto de entrada y el coordinador del flujo del programa. Su función es gestionar la interacción inicial con el usuario y delegar las operaciones a las capas correspondientes.**
* **es.nebrija.dao (Data Access Object): Comprende las clases responsables de la interacción directa con la base de datos MySQL. Cada entidad del modelo de datos cuenta con su propio DAO, lo que centraliza las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y aísla la lógica de acceso a datos del resto de la aplicación. La clase DbConnection.java es crucial en esta capa, ya que gestiona la conexión a la base de datos utilizando el patrón Singleton.**
* **es.nebrija.model: Esta capa define los modelos de datos que representan las entidades del dominio del problema (Personaje, Usuario, Mochila, Objeto). Además, incluye clases de utilidad como Sesion.java (para la gestión de usuarios), Eleccion.java (para la validación de entrada de usuario) y LanzadorLore.java (para la presentación secuencial de texto).**

3. Análisis y Diseño del Sistema

**Para asegurar una comprensión clara de la estructura y el comportamiento de "ETPA 2", se llevaron a cabo varias fases de análisis y diseño, plasmadas en los siguientes diagramas:**

## 3.1. Diagrama de Flujo Narrativo del Juego

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**El primer diagrama desarrollado es un diagrama de flujo que representa la estructura narrativa y de toma de decisiones del juego. Este esquema define de forma visual el comportamiento del jugador dentro de la historia, así como la evolución del argumento según las decisiones que se vayan tomando.**

**El flujo comienza con la acción "Iniciar partida", que marca el punto de entrada al sistema interactivo. A continuación, se permite al jugador "Elegir personaje", lo cual influye directamente en el "Tipo de historia" que se le asignará. Esta lógica está basada en que diferentes personajes pueden tener distintos trasfondos, habilidades u objetivos, lo que da lugar a rutas narrativas divergentes.**

**La estructura considera además condiciones que alteran el entorno o la experiencia del jugador, como los "Efectos del tiempo" (por ejemplo, tormentas, niebla o cambios de hora), y mecánicas propias del *gameplay* como "Conseguir objetos", que pueden ser necesarios para progresar o desbloquear nuevas interacciones.**

**El juego incluye decisiones de combate, donde el jugador debe "Elegir si luchar o no". Si elige luchar, entra en un sistema de combate simplificado, en el cual se simula azar mediante la acción de "Elegir dado", representando una mecánica de probabilidad similar a los juegos de rol. Posteriormente, se permite la acción de "Atacar".**

**Todos estos caminos confluyen en un nodo central denominado "Continuación de historia", donde se consolidan los cambios provocados por decisiones y eventos. Finalmente, el diagrama concluye con la lógica de "Según elección una final u otra", que establece que el juego puede finalizar de diferentes formas, dependiendo del recorrido del jugador, fomentando así la rejugabilidad.**

**Este diagrama permite visualizar la estructura narrativa de forma clara y controlada, asegurando que todas las decisiones estén conectadas y tengan consecuencias lógicas.**

## 3.2. Diagrama de Clases

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**El segundo diagrama corresponde a un diagrama de clases UML que modela la estructura orientada a objetos del sistema. Este modelo es clave para el diseño del código fuente, ya que representa de forma precisa las entidades que forman parte del programa, así como las relaciones entre ellas.**

**Cada clase se representa como un rectángulo dividido en tres partes: el nombre de la clase, sus atributos (propiedades) y sus métodos (funciones). Por ejemplo, una clase Personaje podría incluir atributos como nombre, nivel y vida, así como métodos como atacar(), recibirDaño() o subirNivel().**

**Las relaciones entre clases se representan mediante líneas con distintas notaciones, como herencia (flecha con punta cerrada) o asociación. Estas conexiones indican cómo colaboran las clases entre sí. Por ejemplo, un Personaje podría tener una relación de composición con una clase Inventario, indicando que cada personaje contiene un conjunto de objetos.**

**El uso del diagrama de clases permite organizar el proyecto de forma modular, favoreciendo el mantenimiento, la reutilización del código y la escalabilidad del sistema. Además, este tipo de diseño es útil para aplicar principios de la programación orientada a objetos, como encapsulamiento, herencia y polimorfismo.**

## 3.3. Diagrama Entidad-Relación (E-R)

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**El tercer diagrama utilizado es un diagrama entidad-relación, cuyo objetivo es representar la estructura lógica de la base de datos del juego. Este modelo permite visualizar las entidades principales del sistema y cómo se relacionan entre sí, siendo un paso previo fundamental antes de crear las tablas reales en un sistema gestor de bases de datos (SGBD).**

**Cada entidad está representada mediante un rectángulo, donde se indican los atributos relevantes, incluyendo la clave primaria (PK), que sirve como identificador único de cada fila. Por ejemplo, una entidad Jugador podría tener los atributos id\_jugador, nombre, y fecha\_registro.**

**Las relaciones entre entidades se muestran mediante líneas que conectan las entidades implicadas, acompañadas por los cardinales que indican la naturaleza de la relación: uno a uno (1:1), uno a muchos (1:N) o muchos a muchos (N:M). Estas relaciones son fundamentales para mantener la integridad referencial del sistema.**

**Además, el diagrama ayuda a identificar posibles errores de normalización y redundancia de datos. En este caso, se ha buscado una estructura normalizada, en la que cada dato se almacena una única vez, lo que mejora el rendimiento y la coherencia del sistema.**

**Este diseño servirá como base para generar las instrucciones SQL necesarias para la creación de tablas, relaciones mediante claves foráneas, y posibles restricciones como NOT NULL o UNIQUE.**

4. Componentes Clave y su Funcionalidad

**Aquí se detallan los componentes más relevantes implementados en "ETPA 2" y su papel dentro de la arquitectura del proyecto.**

## 4.1. Main.java

**Como clase principal, Main.java orquesta el inicio de la aplicación. Presenta un menú inicial al usuario para optar entre iniciar sesión o registrarse, delegando la gestión de estas operaciones a la clase Sesion. La utilización de Eleccion.elegido() para validar la entrada del usuario demuestra una preocupación por la robustez de la aplicación desde el punto de inicio.**

## 4.2. DbConnection.java

**Esta clase implementa el patrón de diseño Singleton para garantizar que solo exista una instancia de la conexión a la base de datos a lo largo de toda la aplicación. Esto optimiza el uso de recursos y previene posibles conflictos de conexión. La configuración de la conexión (URL, usuario, contraseña) se define como constantes para una fácil modificación.**

## 4.3. Clases DAO

**Cada clase DAO (e.g., DaoUsuario.java, DaoMochila.java, DaoObjeto.java, etc.) encapsula la lógica de acceso a datos para una entidad específica. Esto incluye métodos para:**

* **selectAll(): Recuperar y mostrar todos los registros de la tabla correspondiente.**
* **cargarX(): (En DaoMochila.java, DaoObjeto.java) Recuperar una lista de objetos desde la base de datos.**
* **guardarX(): Insertar un nuevo registro en la base de datos.**
* **actualizarX(): Modificar un registro existente.**
* **eliminarX(): Borrar un registro.**
* **getUsuario(): (En DaoUsuario.java) Autenticar un usuario por sus credenciales.**
* **registrarUsuario(): (En DaoUsuario.java) Insertar un nuevo usuario en la base de datos.**

**La gestión de excepciones SQLException en cada DAO asegura un manejo básico de errores relacionados con la base de datos.**

## 4.4. Clases Modelo

**Las clases dentro de es.nebrija.model representan la estructura de los datos del juego, directamente inspiradas y modeladas a partir del diagrama de clases UML y el diagrama entidad-relación, lo que asegura una correspondencia clara entre el diseño lógico y la implementación del código.**

* **Usuario.java: Define las propiedades de un usuario (ID, nombre, email, contraseña).**
* **Mochila.java: Modela una mochila, incluyendo su ID, nombre, descripción, capacidad y valor. Se ha añadido un método mostrarMochilas() para la presentación de los datos.**
* **Objeto.java: Representa un objeto con sus atributos (ID, nombre, descripción, valor, peso). Similar a Mochila, incluye mostrarObjetos() para listar los elementos.**
* **Personaje.java: Define las características de un personaje (ID, nombre, apellidos, raza, edad, ataque, defensa, velocidad).**
* **Sesion.java: Contiene la lógica para inicioSesion() y registroUsuario(). Demuestra la interacción con DaoUsuario y la validación de una clave de registro. La gestión de bucles para reintentos de inicio de sesión mejora la experiencia del usuario.**
* **Eleccion.java: Una clase de utilidad crucial para la interacción con el usuario. Su método estático elegido(int num1, int num2) garantiza que la entrada del usuario sea un número entero dentro de un rango predefinido, previniendo errores de entrada y mejorando la robustez de la aplicación.**
* **LanzadorLore.java: Esta clase proporciona una funcionalidad de presentación narrativa mediante el método estático lanzador(ArrayList<String> historia). Permite mostrar secuencias de texto con un retardo, lo cual es útil para introducir elementos de historia o tutoriales de manera atractiva, alineándose con la progresión narrativa definida en el diagrama de flujo.**
* **Batalla.java: Esta clase gestiona la lógica de las batallas dentro del juego. Ofrece al usuario la opción de elegir entre dos tipos de batalla ("Valle de Hierro" o "Montaña de Fuego"). Genera un clima aleatorio para la batalla, aplicando un efecto específico en función del clima (por ejemplo, reducción de velocidad con "Lluvia helada" o disminución de ataque con "Vendaval arenoso"). Además, utiliza LanzadorLore.historiaJorge() para presentar la narrativa asociada a la batalla elegida.**

5. Configuración de la Base de Datos

**El sistema se conecta a una base de datos MySQL local (localhost:3306) denominada etpa2. Las credenciales de acceso son root para el usuario y root para la contraseña. Es fundamental que el entorno disponga de una instancia de MySQL con esta configuración y que las tablas correspondientes a las entidades (Aliado, Enemigo, Mochila, Objeto, Personaje, Usuario) estén correctamente definidas para el funcionamiento del sistema, estructuras que fueron diseñadas y validadas mediante el diagrama entidad-relación.**

6. Funcionalidades Implementadas

**El proyecto "ETPA 2" ofrece las siguientes funcionalidades principales:**

* **Gestión de Autenticación: Permite a los usuarios iniciar sesión y registrarse, con una validación básica de credenciales y una clave de registro para nuevos usuarios.**
* **Persistencia de Datos: A través de las clases DAO, la aplicación interactúa con la base de datos para almacenar, recuperar, actualizar y eliminar información sobre usuarios, personajes, mochilas y objetos, basándose en el modelo de datos definido en el diagrama entidad-relación y el diagrama de clases.**
* **Listado de Entidades: Funcionalidades para mostrar en consola los datos de aliados, enemigos, mochilas y objetos, recuperados directamente de la base de datos.**
* **Control de Entrada del Usuario: La implementación de Eleccion.java demuestra la capacidad de crear utilidades para validar la entrada del usuario, mejorando la fiabilidad de la aplicación.**
* **Presentación Narrativa: La clase LanzadorLore.java introduce un elemento de narrativa básica, permitiendo una presentación de texto secuencial con temporización, lo cual soporta la progresión de la historia y las ramificaciones definidas en el diagrama de flujo narrativo.**

7. Lecciones Aprendidas y Posibles Mejoras Futuras del Proyecto

**Como equipo, el desarrollo de "ETPA 2" nos ha proporcionado una valiosa oportunidad para aplicar y consolidar los conocimientos adquiridos en 1º de DAM, especialmente en el diseño de una arquitectura de software basada en capas y la interacción con bases de datos. Si bien hemos logrado los objetivos principales de este proyecto, durante la implementación y revisión, hemos identificado varias áreas clave donde podríamos mejorar y expandir el sistema en futuras fases. Estas posibles mejoras demuestran nuestra comprensión de los retos inherentes al desarrollo de software y nuestro compromiso con la creación de soluciones más robustas y profesionales.**

**A continuación, detallamos las áreas de mejora que consideramos prioritarias:**

**Seguridad y Gestión de Credenciales**

* **Cifrado de Contraseñas: Una de las mejoras más críticas que hemos detectado es la forma en que almacenamos las contraseñas de los usuarios. Actualmente, se guardan en texto plano en la base de datos. En el futuro, sería fundamental implementar un algoritmo de hashing seguro (como SHA-256 o bcrypt) junto con "salts" para proteger las contraseñas, incluso si la base de datos se viera comprometida. Esto es una práctica estándar de la industria que debemos adoptar.**
* **Manejo de la Clave de Registro: La clave de registro ("JuegosReunidos123") está "hardcodeada" en el código. Para una mayor seguridad, deberíamos externalizar esta clave (por ejemplo, a un archivo de configuración que no se incluya en el control de versiones) o implementar un mecanismo de generación o validación más dinámico.**

**Manejo de Excepciones y Logging**

* **Mensajes de Error Detallados y Consistentes: Si bien hemos capturado las excepciones SQL, los mensajes mostrados al usuario a veces son genéricos o simplemente indican un "Error". Podríamos mejorar la experiencia de usuario ofreciendo mensajes más específicos que ayuden a identificar el problema, y para nosotros como desarrolladores, implementar un sistema de logging (por ejemplo, con librerías como Log4j) para registrar los errores de manera estructurada con su pila de llamadas, lo que facilitaría enormemente la depuración.**
* **Validación de Entrada Rigurosa: Aunque Eleccion.java nos ayuda a validar rangos numéricos, la validación de otros tipos de entrada (como formatos de correo electrónico, longitudes máximas de cadenas, o valores específicos para atributos de personaje) no siempre es exhaustiva. Es crucial que la capa de lógica de negocio (o incluso el propio modelo) valide los datos antes de que lleguen a la base de datos para prevenir inconsistencias y errores.**

**Abstracción y Reutilización de DAOs**

* **DAOs Genéricos: Hemos notado cierta repetición de código en nuestros DAOs, especialmente en métodos comunes como selectAll() que imprimen directamente en consola. Una mejora arquitectónica importante sería la creación de una interfaz o clase abstracta GenericDao<T> que defina las operaciones CRUD básicas, permitiendo a los DAOs específicos extenderla y reducir la duplicidad. Esto nos permitiría reutilizar código y mantener una mayor coherencia.**
* **Devolución de Listas en Lugar de Impresión Directa: Los métodos selectAll() en varios DAOs imprimen los resultados directamente. Para una mayor flexibilidad y separación de responsabilidades, estos métodos deberían devolver listas de objetos (ArrayList<Entidad>). De esta manera, la capa que llama a estos métodos (por ejemplo, el controlador) podría decidir cómo presentar esos datos al usuario, ya sea en consola, una GUI o cualquier otro formato.**

**Mejora de la Interfaz de Usuario (UI) y Experiencia de Usuario (UX)**

* **Actualmente, la interacción se realiza a través de la consola de texto. Para el futuro, nos gustaría explorar la implementación de una Interfaz Gráfica de Usuario (GUI). Esto podría realizarse con tecnologías como Java Swing o JavaFX, lo que proporcionaría una experiencia mucho más intuitiva, visual y profesional para el usuario del juego. Esto sería un gran paso para hacer el "juego" más interactivo y accesible.**

**Gestión de Transacciones en la Base de Datos**

* **En operaciones que implican múltiples pasos que modifican la base de datos (por ejemplo, al registrar un usuario y, al mismo tiempo, crear su mochila inicial), si uno de esos pasos falla, los datos podrían quedar inconsistentes. Implementar transacciones JDBC nos permitiría asegurar que todas las operaciones se completen con éxito o, en caso de fallo, que todas se reviertan, garantizando la integridad y consistencia de los datos.**

**Estas posibles mejoras reflejan nuestra visión a largo plazo para el proyecto y nuestro deseo de seguir profundizando en las mejores prácticas de desarrollo de software, aplicando lo aprendido en el aula a situaciones más complejas.**

8. Conclusión

**El proyecto "ETPA 2" es un sólido ejercicio de aplicación de los conocimientos adquiridos en el primer curso de DAM, demostrando la capacidad de diseñar y desarrollar una aplicación Java con interacción de base de datos. La implementación de la arquitectura DAO y la incorporación de utilidades para la interacción y presentación de información reflejan una comprensión de los principios de diseño de software. Los diagramas de flujo narrativo, UML y entidad-relación fueron herramientas fundamentales que guiaron la conceptualización y la implementación del sistema, asegurando una estructura lógica y modular. Este proyecto sienta las bases para futuras exploraciones en el desarrollo de aplicaciones más complejas y robustas.**

**Principales cambios y por qué:**

* **Nueva sección "Análisis y Diseño del Sistema" (Sección 3): Esta es la clave. Creé una nueva sección para agrupar las descripciones de tus tres diagramas. Esto le da a tu memoria una estructura lógica y profesional.**
* **Introducción: Añadí una frase en la introducción para mencionar brevemente que se usaron diagramas para el diseño del sistema, creando una expectativa.**
* **Sección 4.4 (Clases Modelo): Hice una pequeña adición para conectar las clases modelo con los diagramas UML y E-R, mostrando cómo el diseño se traduce en código.**
* **Sección 5 (Configuración de la Base de Datos): Añadí una mención a que la configuración se basó en el diagrama E-R.**
* **Sección 6 (Funcionalidades Implementadas): Conecté la funcionalidad de persistencia de datos y la presentación narrativa con los diagramas correspondientes.**
* **Conclusión: Reforcé el papel de los diagramas como herramientas fundamentales en el diseño del proyecto.**
* **Tono consistente: Mantuve el tono profesional y colaborativo que ya tenías.**